(29) Using Python for accessing Web data

Univ of Michigan
Chapter 11

Regular expressions

- In computing, a regular expression ("regex" or "regexp") provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor.
- Clever "wild card" expressions for matching and parsing strings.

Understanding Regular expressions

- Very powerful and quite cryptic
- Fun once you understand them
- Regular expressions are a language unto themselves
- A language of 'marker characters'.
- Kind of "old school" language - compact

Regular expressions quick guide

^ : matches the beginning of a line
$: matches the end of the line
. : matches any character
\s : matches whitespace
\S : matches any non-whitespace character
* : Repeats a character zero or more times (greedy)
*? : Repeats a character zero or more times (non-greedy)

Using Python to access Web data

Univ of Michi
Chapter 11

+ : Repeats a character one or more times
+? : Repeats a character one or more times (non-greedy)
[aeiou] : matches a single character in the listed set
[^XYZ] : matches a single character not in the listed set
[a-z0-9] : The set of characters can include a range
(: Indicates where string extraction is to start
) : Indicates where string extraction is to end

The regular expression module

- Before you can use regular expressions in your program, you must import the library using "import re"
- You can use re.search() to see if a string matches a regular expression, similar to using the find() method for strings.
- You can use re.findall() extract portions of a string that match your regular expression similar to a combination of find and slicing: var[5:10]

Using re.search() like find()

```
hand = open('file.txt')
for line in hand:
    line = line.rstrip()
    if line.find('From:') >= 0
        print line
```

```
import re
hand = open('file.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:',
    line):
    print line
```

(31) Using Python to access Web data

chapter 11

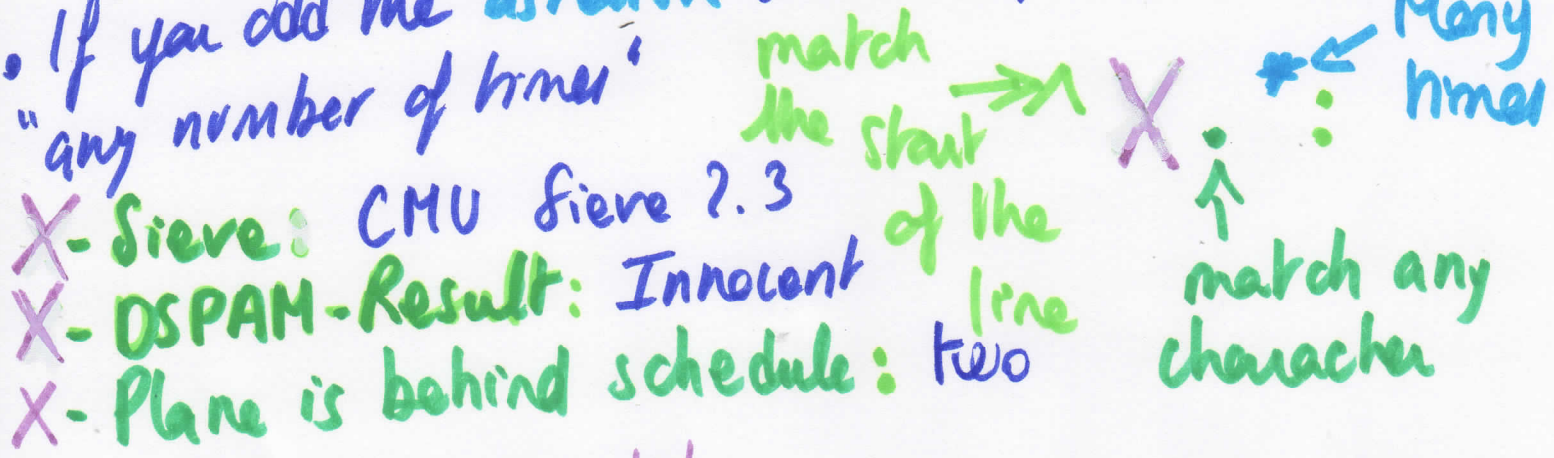
Using re.search() like startswith()

We find-tune what is matched by adding special characters to the string (see previous code)

`line.startswith('From:')` `re.search('^From:', line)`

Wild-card characters

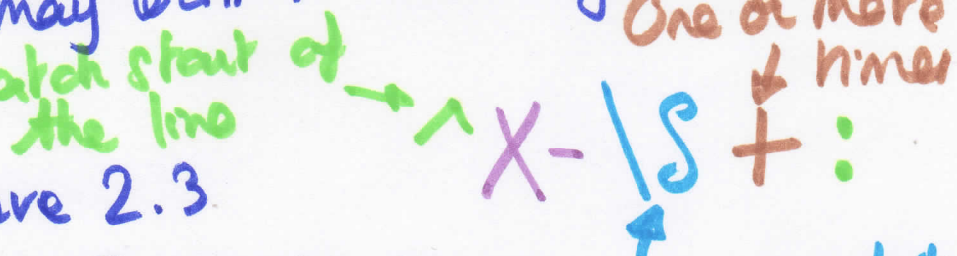
- The dot character matches any character
- If you add the asterisk character, the character is "any number of times"



X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-Plane is behind schedule: two

Fine-tuning your match

Depending on how "clean" your data is and the purpose of your application, you may want to narrow your match down a bit.



X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-Plane is behind schedule: two

Matching and extracting data

- The `re.search()` returns a True/False depending on whether the string matches the regular expression

(32) Using Python to access web data

Univ of Michi
Chapter 11

- If we actually want the matching strings to be extracted, we use re.findall(). It returns a list of zero or more sub-strings that match the regular expression.

```
> import re
> x = 'My 2 favorite numbers are 19 and 42'
> y = re.findall('[0-9]+', x)
> print y
['2', '19', '42']
> y = re.findall('[AEIOU]+', x)
> print y
[]
```

one or more digits

Warning: greedy matching

- The repeat characters (* and +) push outward in both directions (greedy) to match the largest possible string.

```
> import re
> x = 'From: Using the : characters'
> y = re.findall('^F.+:', x)
> print y
['From: Using the :']
```

One or more characters

last char in the match is a :

^F.+:

First character in the match is an F

Why not 'From:'? non greedy

Non-greedy matching

- Not all regular expression repeat codes are greedy! If you add a ? character.

^F.+?:

One or more characters but not greedy

Fine-tuning string extraction

- You can refine the match for re.findall() and separately determine which portion of the match is to be extracted by using parentheses.
- Parentheses are not part of the match, but they tell where to start and stop what string to extract.

```
From stephen.marquard@uct.ac.za Sat Jan ...
> y = re.findall('\S+@\S+', x)
> print x ['stephen.marquard@uct.ac.za']
> y = re.findall('^From:.*? (\S+@\S+)', x)
> print y ['stephen.marquard@uct.ac.za']
```

^From: (\S+@\S+) ← start and end of extract

At least one non-whitespace character

Escape character

- If you want a special regular expression character to just behave normally (most of the time) you prefix it with '\'

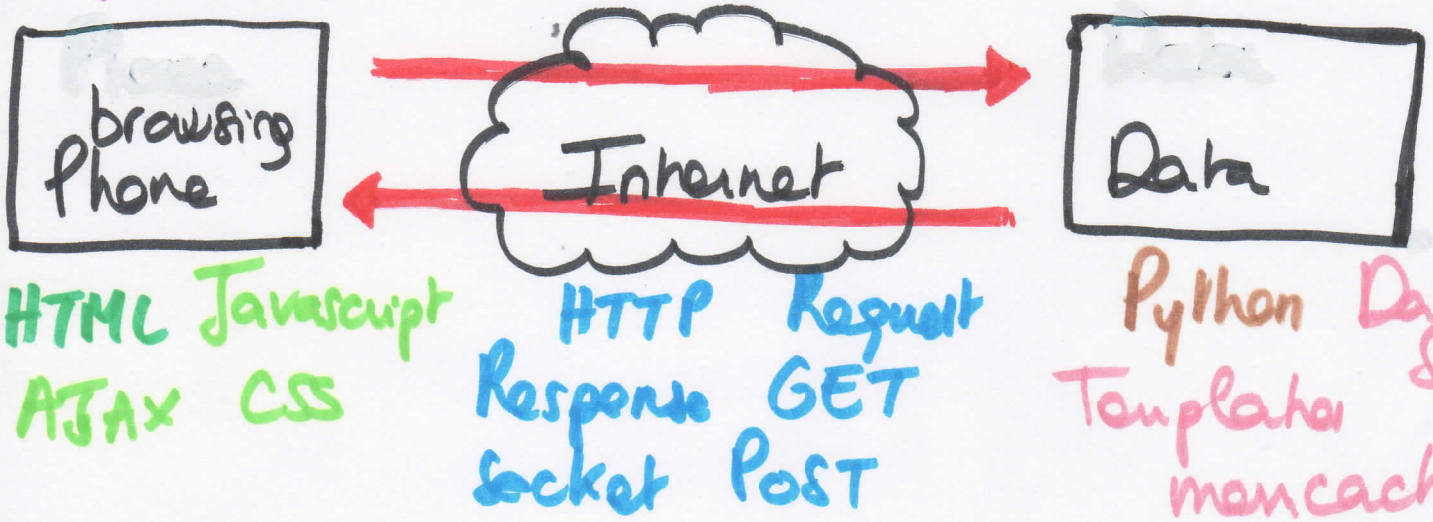
E.g. \$ ← real dollar sign

Summary regular expressions

- Regular expressions are a cryptic but powerful language for matching strings and extracting elements from those strings. See pdf lecture for more examples

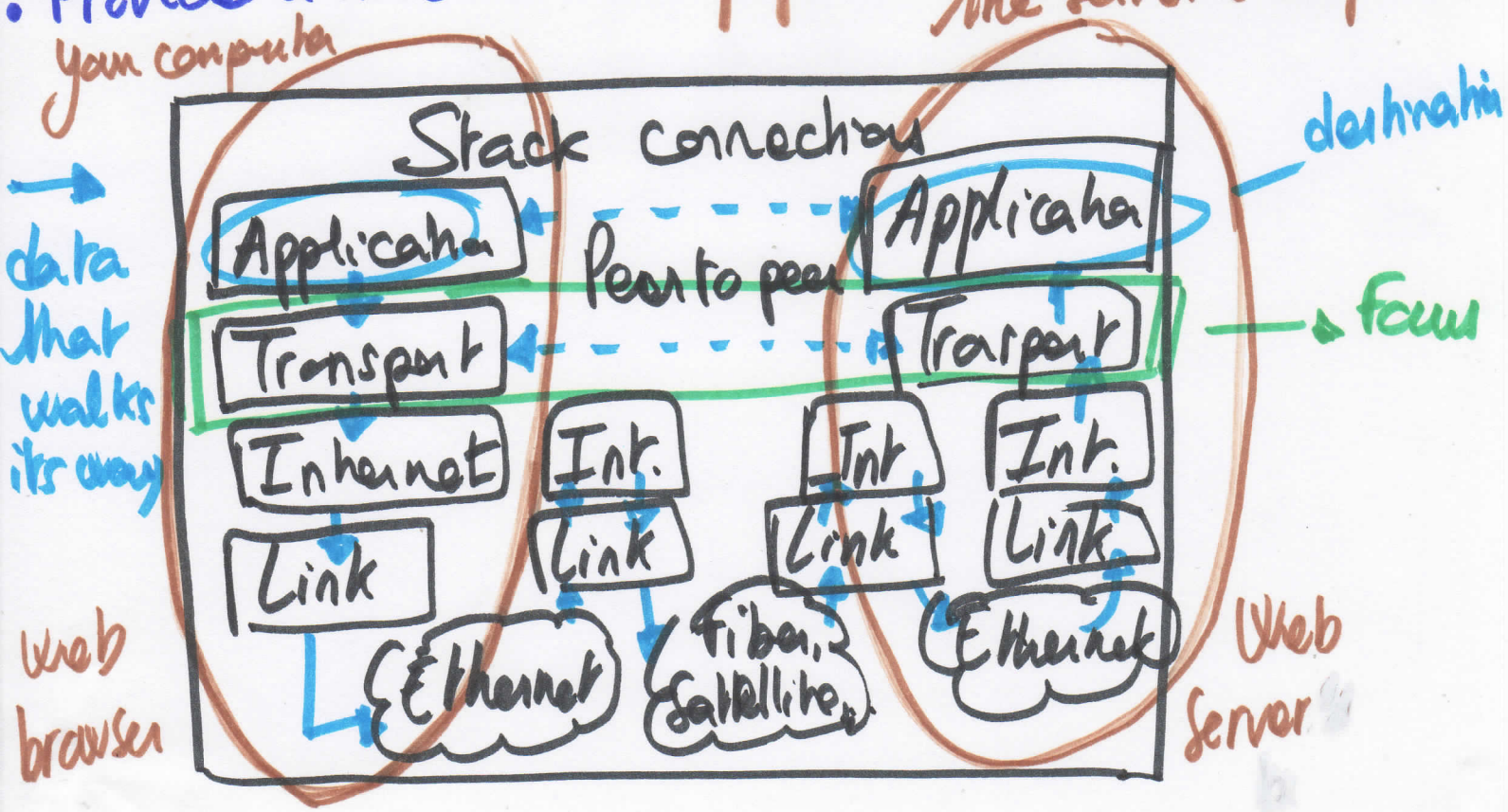
(34) Using Python to access web data Uni of Michi
Chapter 12

Response - request - networked program



Transport Control Protocol (TCP) - Network Architecture

- Built on top of IP (Internet protocol)
- Assumes IP might lose some data - stores and retrans.
mits data if it seems to be lost
- Handles "flow control" using a transmit window
- Provide a nice reliable pipe

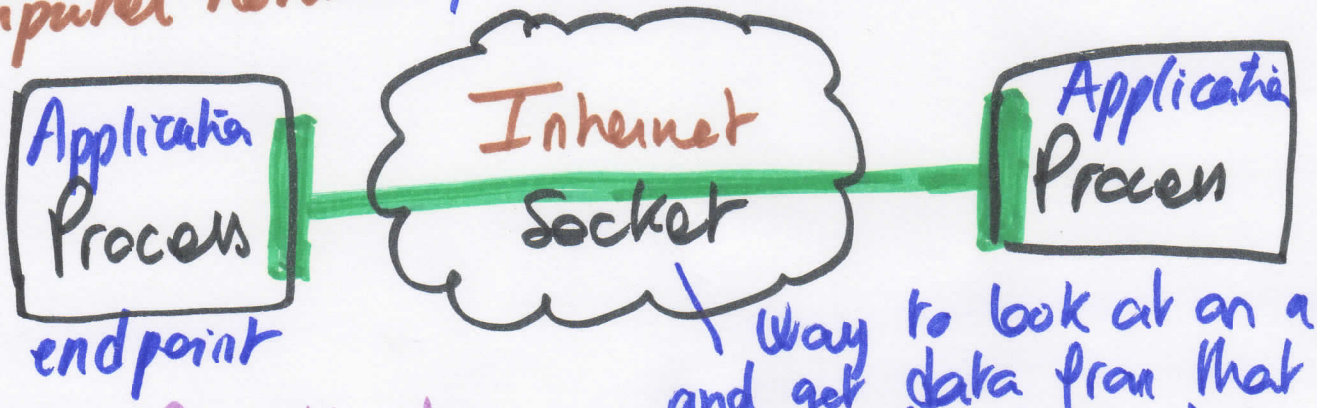


(35) # Using Python to access web data

Uni of Mich
chapter 12

TCP connections / sockets

- A socket is two computer applications / processes that are talking across a network
- "In computer networking, an internet socket or network socket is an endpoint of a bidirectional inter-process communication flow across an Internet Protocol-based computer network, such as the internet."



TCP Port Number

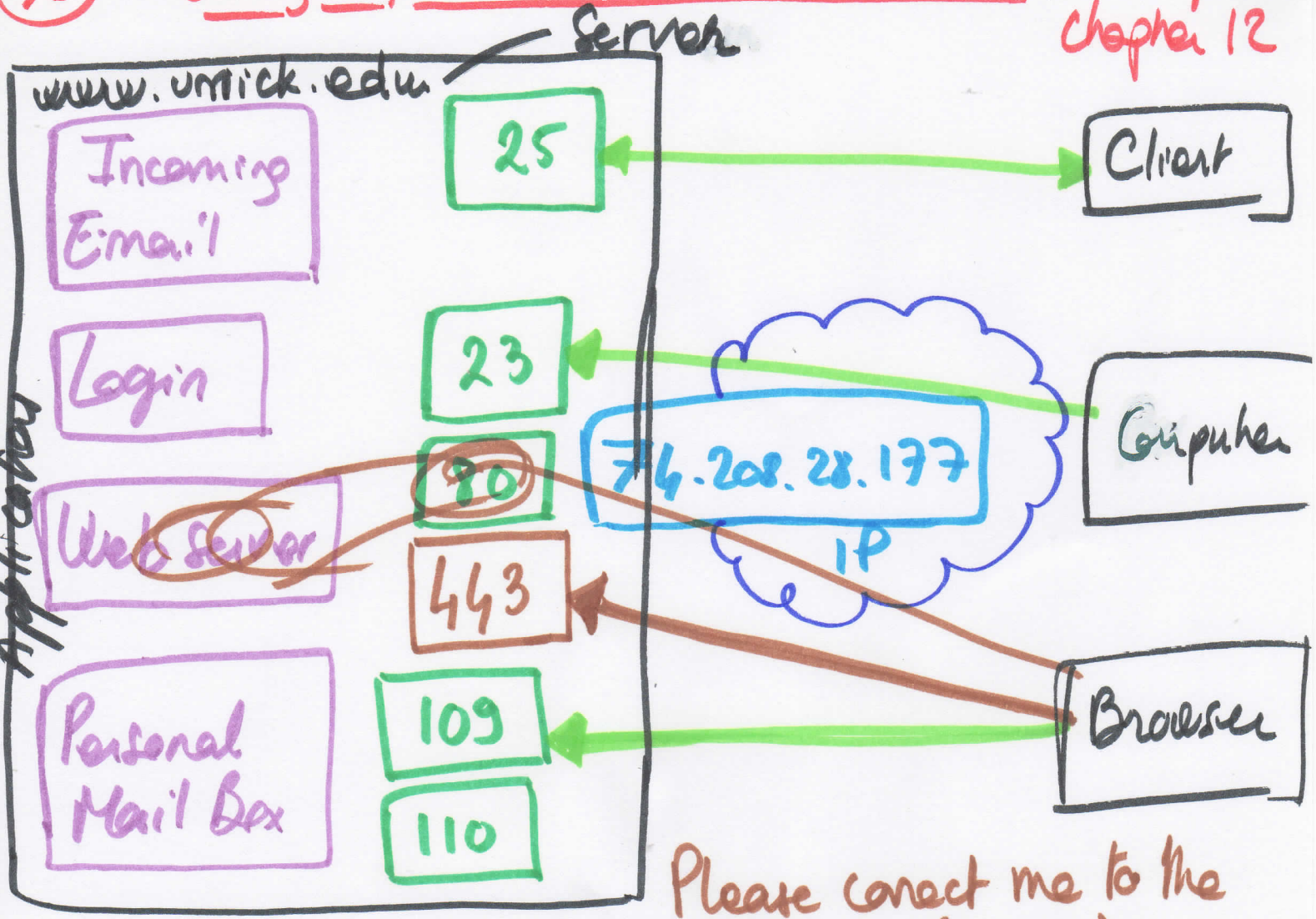
- A port is an application-specific or process-specific software communication endpoint.
- It allows multiple networked applications to coexist on the same server.
- There is a list of well-known TCP port numbers

Common TCP Ports

- Telnet (23) - login
- SSH (22) - Secure login
- HTTP (80)
- HTTPS (443) - Secure
- SMTP (25) (Mail)
- IMAP (143/220/993) - Mail retrieval
- POP (109/110) - Mail retrieval
- DNS (53) - Domain name
- FTP (21) | File transfer

(36) # Using Python to access Web data

Univ of Michi
Chapter 12



Please connect me to the web server (port 80) on http://www.dr-chuck.com

- Sometimes we see the port number in the URL if the web server is running on a "non-standard" port. http ≠ the 80 for
  http://localhost:8085/sikai.htm

Sockets in Python

- Python has built-in support for TCP Sockets (library)

```
> import socket                                  # library
> mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)   # method within library; open a port; half
> mysock.connect(('py4info.com', 80))            # HOST, PORT → establish a connection to an appli
```

(37) Using Python to access Web data

Univ of Mich
Chapter 12

Application protocol

- Since TCP (and Python) gives us a reliable socket, what do we want to do with the socket? What problem do we want to solve? → See Stacks connection previous
- Application protocols: Mail / World Wide Web

HTTP - HyperText Transport Protocol

- The dominant Application Layer Protocol on the Internet
- Invented for the Web - to retrieve HTML; Images, Docs...
- Extended to be data in addition to documents - RSS, Web services, etc...
- Basic concept - Make a connection - Request a document - Retrieve the document - close the connection

Very simple protocol used for numerous things!

HTTP: The HyperText Transport Protocol is the set of rules to allow browsers to retrieve web documents from servers over the internet.

What is a protocol?

- A set of rules that all parties follow so we can predict each other's behavior
- And not bump into each other (like driving rules)

URL

what protocol are we going to use to talk to the server | what data are we going to get?

http://www.dr-chuck.com/page1.htm

protocol | What host are we going to talk to | document

Using Python to access Web data

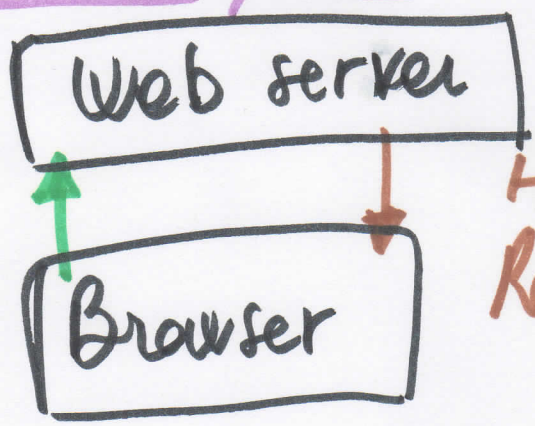
Uni of Michi
Chapter 12

Getting data from the server

- Each time the user clicks on an anchor tag with an `href=` value to switch to a new page, the browser makes a connection to the web server and issues a "GET" request - to GET the content of the page at the specified URL.
- The server returns the HTML doc to the browser, which formats and displays the doc to the user.

Request-response cycle: at the moment you click, it makes the request, the other page it gets is the response

Making an HTTP request



Internet standards

- The standards for all of the internet protocols (inner workings) are developed by an organization.
- Internet Engineering Task Force (IETF)
- www.ietf.org
- Standards are called "RFCs" - "Request for Comments"

(39) # Using Python to access Web data

Uni of Michi
chapter 12

A browser debugger reveals detail...

- Most browsers have a developer mode so you can watch it in action
- It can help the HTTP request-response cycle
- Some simple-looking pages involve lots of requests: HTML page(s), image files, CSS Style sheets, JavaScript file

An HTTP request in Python

```
> import socket
> mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
> mysock.connect(('www.py4inf.com', 80))
> mysock.send('GET http://www.py4inf.com/code/romeo.txt HTTP/1.0\n\n')
> while True:
    data = mysock.recv(512)
    if (len(data) < 1):
        break
    print data
> mysock.close()
```

create the endpoint

pick the endpoint to the web make a connection

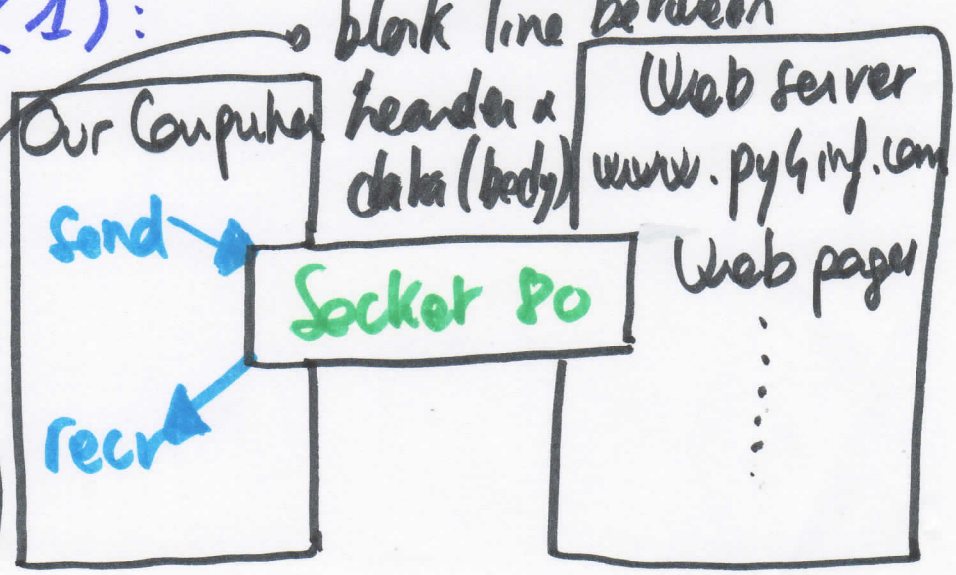
We now have a socket

characters (up to)

blank line between headers & data (body)

you get Headers (meta-data) and body (up to 512 ch)

Very simple web browser



(40) Using Python to access web data

Chapter 12

Making HTTP easier with urllib

- Since HTTP is so common, we have a library that does all the socket work for us and makes web pages look like a file.

— same as socket but shorter with urllib you only get the data (ugly) (code/romeo.txt)

```
> import urllib
> fhand = urllib.urlopen('http://www.py4inf.com/
```

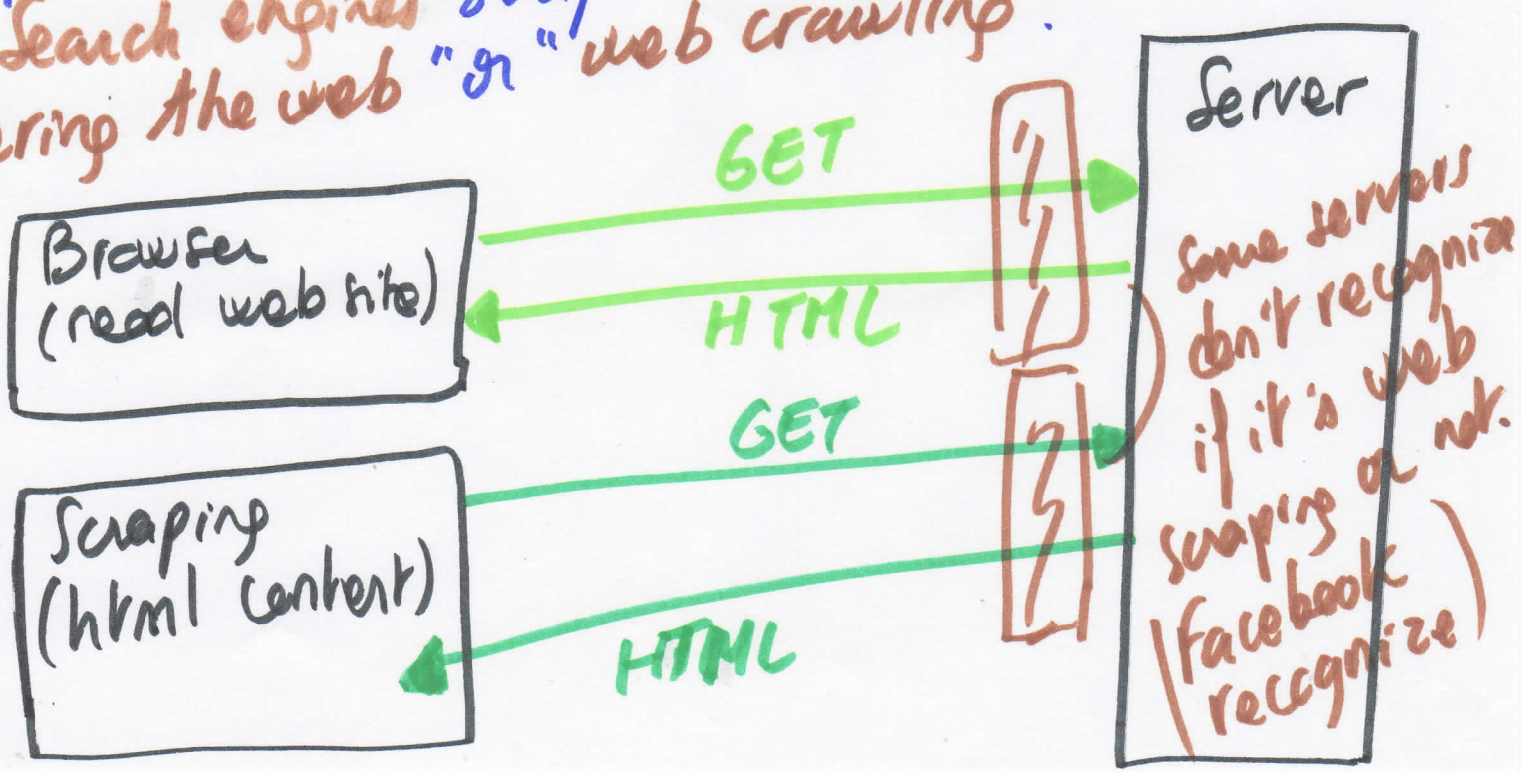
and then you can write pretty much any program...

```
> for line in fhand:
    print line.strip()
```

) same as when we were reading files

Parsing HTML - a.k.a Web scraping

- When a program or script pretends to be a browser and retrieves web pages, looks at those web pages, extracts information, and then looks at more web pages.
- Search engines scrape web pages - we call this "spidering the web" or "web crawling".



(41) Using Python to access web data

Univ. of Mich
Chapter 12

Why scrape?

- Pull data - particularly social data - who links to who?
- Get your own data back out of some system that has no "export capability"
- Monitor a site for new information ⚠ don't create a search engine
- Spider the web to make a database for a search engine

Scraping web pages

- There is some controversy about web page scraping and some sites are a bit snippy about it (Google, facebook)
- Republishing copyrighted information is not allowed
- Violating terms of service is not allowed

The easy way to scrap the web - Beautiful Soup

- You could do string searches the hard way
- Or use the free software called BeautifulSoup from www.crummy.com
- Place the BeautifulSoup.py file in the same folder as your Python code.

```
>import urllib
>from BeautifulSoup import *
>url = raw_input('Enter-')
>
```

(42) Using Python to access Web data

uni of Michi
Chapter 13

```
html = urllib.urlopen(url).read()   # read it all (web page) a string
soup = BeautifulSoup(html)
# soup is a soup object.
# Retrieve a list of the anchor tags
# Each tag is like a dictionary of HTML
attributes
tags = soup('a')
for tag in tags:
    print tag.get('href', None)     # if there is no href
```

soup object = parsed html data and you can ask soup questions.

Summary HTTP

- The TCP/IP gives us pipes/sockets between applications
- We designed application protocols to make use of these pipes.
- HTTP is a simple yet powerful protocol
- Python has good support for sockets, HTTP, and HTML parsing

Data on the Web

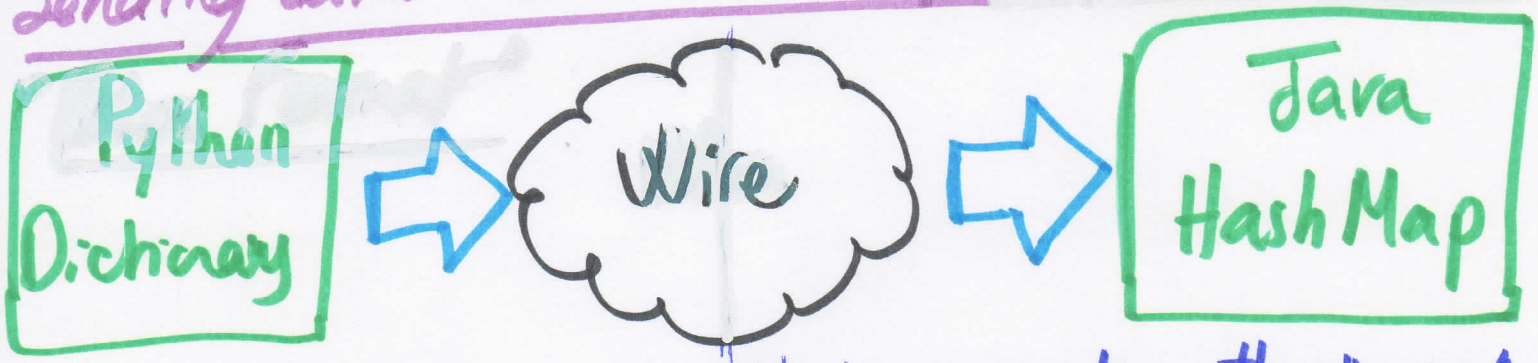
- With the HTTP Request/Response well understood and well supported, there was a natural move toward exchanging data between programs using these protocols.
- We needed to come up with an agreed way to represent data going between applications and across networks

(43) # Using Python to access Web data

Uni of Michi chapter 13

- There are two commonly used formats:
  XML and JSON

Sending data across the "Net"



a.k.a "Wire protocol" - What we send on the "wire"
Example with a python dictionary

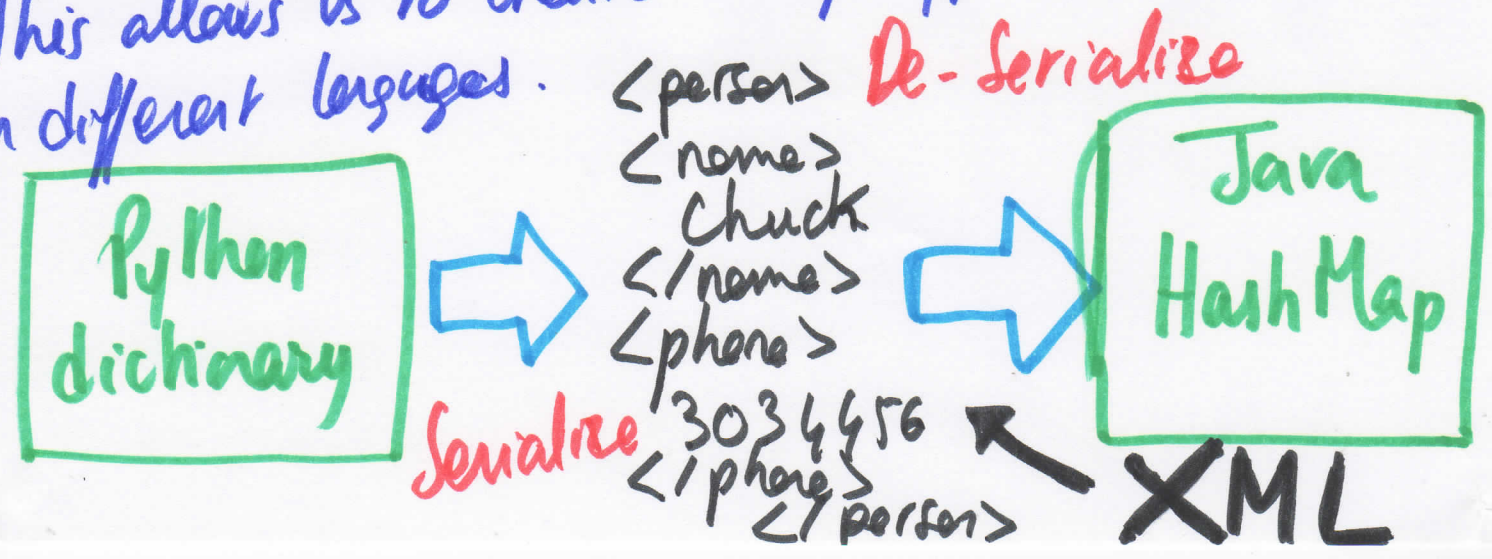
Agreeing on a "Wire Format"

Wire format: (here) neither Python nor Java.

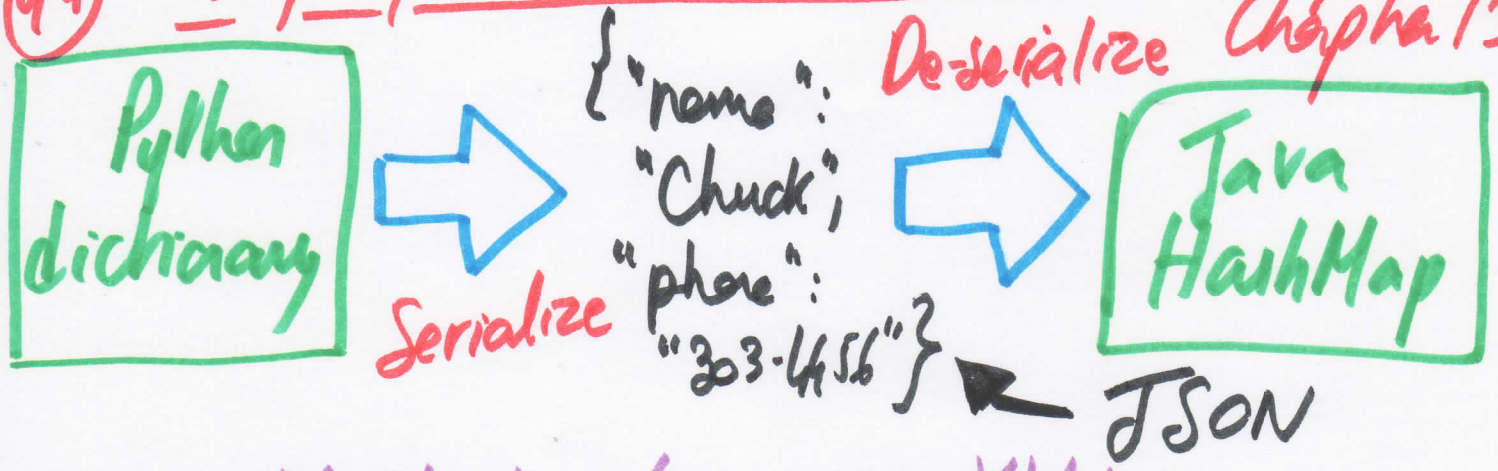
Serialize: act of taking internal structure and create a wire format

De-serialize: act of taking that wire format and create internal structure in a different language.

- This allows us to create set of applications that work in different languages.



(44) Using Python to access web data — Univ of Michi — Chapter 13



eXtensible Markup Language - XML

- Marking up data to send across the network
- Primary purpose is to help information systems share structured data.
- It started as a simplified subset of the Standard Generalized Markup Language (SGML) and is designed to be relatively human-legible.

XML "Elements" (or Nodes)

- Simple element
- Complex element

```
<people>
 <person>
  <name> Chuck </name>
  <phone> 303 4456 </phone>
 </person>
 <person>
  <name> Noah </name>
  <phone> 622 7421 </phone>
 </person>
</people>
```

(45) Using Python to access web data

Univ of Michi
Chapter 13

XML basics

- Start tag
- End tag
- Text content
- Attribute
- Self closing tag

```
<person>
  <name> Chuck </name>
  <phone type="intl">
    +1734 303 4456
  </phone>
  <email hide="yes"/>
</person>
```

White space and indentation

- Line ends do not matter.
- White space is generally discarded on text elements.
- We indent only to be readable.
- To get indentation on XML data, use a pretty printer such as http://xml prettyprint.com

XML terminology

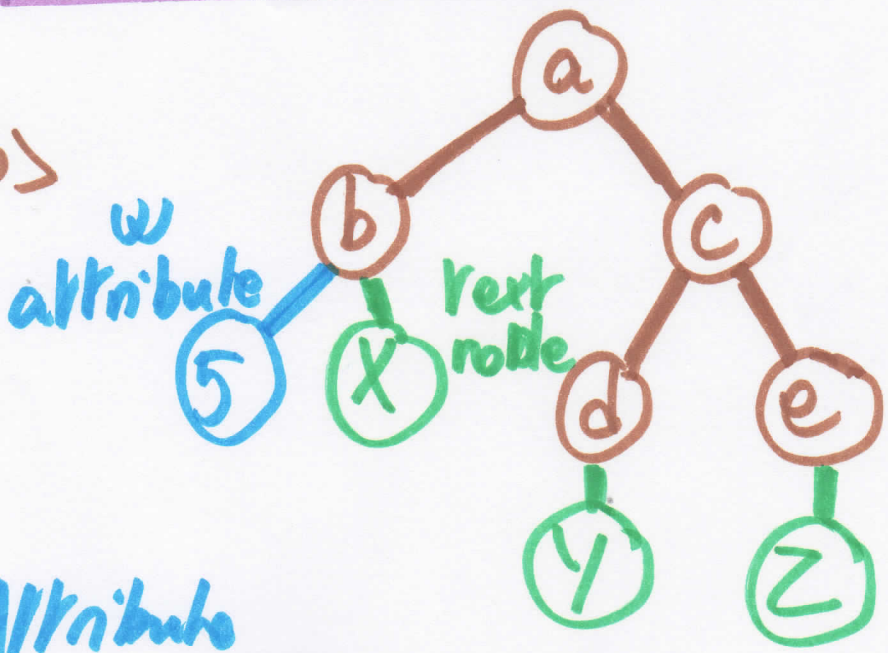
Tags indicate the beginning and ending of elements

Attributes - Keyword/value pairs on the opening tag of XML

Serialize / De-serialize: Convert data in one program into a common format that can be stored and/or transmitted between systems in a programming language-independent manner

XML as a tree (Elements, text and attributes)

```
<a>
 <b w="5">X</b>
 <c>
  <d>Y</d>
  <e>Z</e>
 </c>
</a>
```



Elements Text Attribute

XML as paths

/a/b 5 X
/a/c/d Y
/a/c/e Z

XML Schema

- Describing a "contract" as to what is acceptable XML
- Description of the legal format of an XML document
- Expressed in terms of constraints on the structure and content of documents
- Often used to specify a "contract" between systems - "My system will only accept XML that conforms to this particular Schema."
- If a particular piece of XML meets the specification of the Schema - it is said to "validate"

(47) Using Python to access web data — Univ of Michi

XML document

```
<person>
  <lastname> Severance </lastname>
  ⋮
</person>
```

XML Schema Contract

```
<xs:complexType name="person">
  <xs:sequence>
  ⋮
</xs:complexType>
```

Many XML Schema Languages

- Document Type Definition (DTD)
- Standard Generalized Markup Language
- XML Schema from W3C - (XSD)

XSD XML Schema (W3C spec)

- We will focus on the World Wide Web Consortium (W3C) version
- It is often called "W3C Schema" because "Schema" is considered generic
- More commonly it is called XSD because the file names end in .xsd

XSD structure

- xs:element
- xs:sequence
- xs:complexType

Using Python to access web data


```
<xs:complexType name="person">
  <xs:sequence>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="age" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>
```

XSD Constraints

```
<xs:element name="full_name" type="xs:string"
minOccurs="1" maxOccurs="1"/>
<xs:element name="child_name" type="xs:string"
minOccurs="0" maxOccurs="10"/>
```

```
<full_name>Tove Refsnes</full_name>          ①
<child_name>Hege</child_name>
<child_name>Stale</child_name>               ② (up to 10)
```

XSD Data Types

xs:string	John Smith
xs:date	2002-09-24
type="xs:dateTime"	"2002-05-30T09:30:10Z"
"xs:decimal"	999.50
"xs:integer"	30

ISO 8601 Date/Time Format

2002-05-30T09:30:10Z

- 2002-05-30 → year-month-day
- T09:30:10 → Time of the day
- Z → Time zone - typically specified in UTC/GMT rather than local time zone.

(49) Using Python to access Web data — Univ of Michi

chapter 13

Parsing XML in Python (xml3.py)

XML parsing mechanism

```
import xml.etree.ElementTree as ET
data = '''<person>
  <name> Chuck </name>
  <phone type = "intl">
    +1 734 303 4456
  </phone>
  <email hide = "yes" />
</person>'''
```

one single big string (it could be data get from URL ...)

parsing de-serialization: we get back an object ("tree" here)

```
tree = ET.fromstring(data)
print 'Name:', tree.find('name').text
print 'Attr:', tree.find('email').get('hide')
```

tree.find('name').text → a string

tree.find('email').get('hide')

To use the object "tree" to find stuff in the XML data

```
Name: Chuck
Attr: yes
```

Parsing XML in Python (xml2.py)

```
import xml.etree.ElementTree as ET
input = '''<stuff>
  <users>
    <user x="2">
      <id>001</id>
      <name> Chuck </name>
```

(50) Using Python to access web data

Univ of Michi
Chapter 13

```
</user>
</users>
</stuff> '''
```

like a tree view of the input

find everything that match this path

```
stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print 'User count:', len(lst)
for item in lst:
    print 'Name', item.find('name').text
    print 'Id', item.find('id').text
    print 'Attribute', item.get("x")
```

```
User count: 2        Attribute 2
Name Chuck           Id 001
                                  ...
```

JSON - JavaScript Object Notation

- JSON solves much of the same problems as XML
- However, XML is better when there are a lot of nested elements / the data is complex.
- JSON is less descriptive. It has two basics structures: arrays (like lists) and objects (like dictionaries)
- Advantage: it looks more like Python (syntax)

51 Using Python to access Web data

Uni of Michi chapter 13

JSON example - json1.py (object)

```
import json
data = '''{
  "name" : "Chuck",
  "phone" : {
    "type" : "intl",
    "number" : "+1 734 303 4456"
  },
  "email" : {
    "hide" : "yes"
  }
}'''

info = json.loads(data)
print 'Name:', info["name"]
print 'Hide:', info["email"]["hide"]
```

like a nested "dictionary) an object

key/value pairs followed by comas

key with a value being an object {}

there is no attribute (it has made another value "type")

{ } → a string

loads: load from string de-serialize stop

the output is a dictionary (Python)

info["name"] → Chuck

info["email"]["hide"] → yes

JSON example - json2.py (array)

```
import json
input = '''[
  { "id" : "001",
    "x" : "2",
    "name" : "Chuck"
  }
]'''
```

[ ] → array

one object separated by a comma

[0] id map 001

(52) Using Python to access Web data

Univ of Michi
chapter 13

```
[ {
  "id": "009",
  "x": "7",
  "name": "Chuck"
} ]
```

2nd object

[1]

```
import json
data = '''
[ {
  "id": "009",
  "x": "7",
  "name": "Chuck"
} ]
'''
info = json.loads(input)
print 'User count:', len(info)
for item in info:
    print 'Name', item['name']
    print 'Id', item['id']
    print 'Attribute', item['x']
```

you get a native Python list

Chuck & Chuck

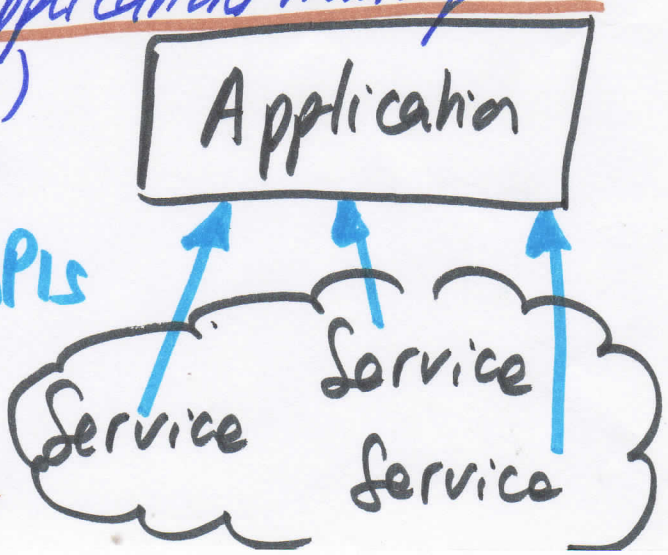
because nested objects

001 & 009

2 & 7

Service oriented approach

- Most non-trivial web application use services
- They use services from other applications (e.g. credit card charge, hotel reservation systems...)
- Services publish the "rules" application must follow to make use of the service (API)
- Applications that work together can take data from many different sources and bring that data together.



(53) Using Python to access Web data

Univ of Mich
chapter 13

Multiple systems

- Initially two systems cooperate and split the problem
- As the data/service becomes useful, multiple applications want to use the information/application.

API - Application Program Interface

- Defined set of rules to interface with an application program
- The API itself is largely abstract in that it specifies an interface and controls the behavior of the objects specified in that interface. The software that provides the functionality described by an API is said to be an "implementation" of the API. An API is typically defined in terms of the programming language used to build an application.

Web service technologies

- SOAP - Simple Object Access Protocol (software)
  - Remote programs / code which we use over the network
  - Too complex, better)
- REST - Representational State Transfer (resource focused)
  - Remote resources which we create, read, update and delete remotely

A pattern that we use

(53) Using Python to access Web data

Univ of Mich
chapter 13

Multiple systems

- Initially two systems cooperate and split the problem
- As the data/service becomes useful, multiple applications want to use the information/application.

API - Application Program Interface

- Defined set of rules to interface with an application program
- The API itself is largely abstract in that it specifies an interface and controls the behavior of the objects specified in that interface. The software that provides the functionality described by an API is said to be an "implementation" of the API. An API is typically defined in terms of the programming language used to build an application.

Web service technologies

- SOAP - Simple Object Access Protocol (software)
  - Remote programs / code which we use over the network
  - Too complex, better)
- REST - Representational State Transfer (resource focused)
  - Remote resources which we create, read, update and delete remotely

A pattern that we use

(54) Using Python to access web data

Univ of Mich
Chapter 13

Google geocoding API - code example (geojson.py)

↳ doesn't require password

```
import urllib   ) retrieve data from url
import json     ) parse data that comes back
serviceurl = 'http://maps.googleapis.com/maps/api/geocode/json?'

while True:
    address = raw_input('Enter location:')
    if len(address) < 1 : break
    url = serviceurl + urllib.urlencode({'sensor':'false',
          'address': address})
    print 'Retrieving', url
    uh = urllib.urlopen(url)
    data = uh.read()
    print 'Retrieved', len(data), 'characters'

    try: js = json.loads(str(data))
    except: js = None
    if 'status' not in js or js['status'] != 'ok':
        print '==== Failure To Retrieve ===='
        print data
        continue

    print json.dumps(js, indent=4)

    lat = js["results"][0]["geometry"]["location"]["lat"]
```

urlencode: to encode this to put it in a url "Do that for me"

read json

dump in the json data and indent it (make it pretty)

pretty pretty

because kitt

Using Python to access web data

Univ of Mich
chapter 13

```
lng = js["results"][0]["geometry"]["location"]["lng"]
print 'lat', lat, 'lng', lng
location = js['results'][0]['formatted_address']
print location
```

user typing

```
Enter location: Ann Arbor, MI
Retrieving http://maps.googleapis.com/...
Retrieved 1669 characters
lat 42.28... lng -83.74...
Ann Arbor, MI, USA
Enter location
```

What is in the string (data):

http://maps.googleapis.com/maps/api/geocode/json?sensor=false&address=Ann+Arbor%2C+MI

```
{ "status": "OK",
  "results": [
    { "geometry": {
        "location_type": "APPROXIMATE",
        "location": {
          "lat": 42.28...
          "lng": -83.74...
        }
```

```
},
"address_components": [
   {
      "long_name": "Ann Arbor",
      "types": [
         "locality",
         "political"
      ],
      "short_name": "Ann Arbor"
   }
],
"formatted_address": "Ann Arbor, MI, USA",
"types": [
   "locality",
   "political"
]
}
]
```

API Security and Rate Limiting

- The computer resources to run these APIs are not "free"
- The data provided by these APIs is usually valuable
- The data providers might limit the number of requests per day, demand an API "key", or even charge for usage
- They might change the rules as things progress....

(57) Using Python to access web data

Uni of Michi chapter 13

Twitter API

- To get access to Twitter API, you need to use your own account and ask an access token (which you can store in a python file for reuse - look hidden.py)
- Twitter uses OAuth (signature system) to secure url
- Look up at the python code & video to know how to do it

Summary - Web services

- Service Oriented Architecture allows an application to be broken into parts and distributed across a network
- An Application Program Interface (API) is a contract for interaction
- Web services provide infrastructure for application cooperating (an API) over a network - SOAP and REST are two styles of web services.
- XML and JSON are serialization formats.